BRAC UNIVERSITY

Inspiring Excellence

Department of Computer Science and Engineering

Course Code: CSE341	**Credits:** 1.5
Course Name: Microprocessors	**Semester:** Spring'25

Lab 07

STACK

I. Topic Overview:

A stack is a one-dimensional data structure that operates on a "Last-In, First-Out (LIFO)" principle. This means that the most recently added item is the first one to be removed. Items are added to and removed from one end of the stack, known as the top of the stack. To store data in the stack, it is "pushed" onto the top. To retrieve data, it is "popped" from the top. Because of its LIFO nature, the last data item pushed into the stack is the first one to be popped out.

The memory space reserved in the stack segment is used to implement the stack. The registers SS and ESP (or SP) are used to implement the stack. The top of the stack, which points to the last data item inserted into the stack, is pointed to by the SS: ESP register, where the SS register points to the beginning of the stack segment, and the SP (or ESP) gives the offset into the stack segment.

II. Lesson Fit:

There is a prerequisite to this lab. Students must have a basic idea of the following concepts:

a. Registers

b. Details of flags

c. Character encoding using ASCII

d. Interrupt in assembly

III. Learning Outcome:

After this lecture, the students will be able to:

a. Perform basic push pop operations using the stack.

b. Solve different problems in a more effective way using a stack.

IV. Anticipated Challenges and Possible Solutions:

a. Students may get confused with PUSH and POP operations, especially when trying to use 8-bit registers (like AL, BL, etc.) as the source for PUSH or the destination for POP. It will show an error.

Solutions:

i. In PUSH/POP operations, data word registers should be used.

V. Acceptance and Evaluation:

Students will show their progress as they complete each problem. They will be marked according to their class performance. There may be students who might not be able to finish all the tasks, they will submit them later and give a viva to get their performance mark. A deduction of 30% marks is applicable for late submission. The marks distribution is as follows:

Code: 50%

Viva: 50%

VI. Activity Detail

A. Hour: 1

Discussion: Basics of the Stack

- The Stack Segment (SS) register holds the starting address of the stack segment in memory. In EMU8086 (16-bit real mode), the stack is accessed using the combination of SS: SP.

  The SS register is primarily used for the following purposes:

- To store temporary data during program execution.
- To store the return address when a subroutine (e.g., a CALL) is invoked.
- To handle the interrupts

- SS (Stack Segment Register): The Stack Segment register, or SS register, stores the starting address of the stack segment in memory.
- SP (Stack Pointer Register): The SP register is initialized based on the value set by the .STACK directive. It points to the top of the stack, and when the program starts, it typically represents the empty position at the end of the stack segment (i.e., the highest address in the reserved stack area).
- When the stack is not empty, SP represents the top of the Stack in order to save the contents of a register during the execution, so that it can be used later for other purposes.
- To do so, the microprocessor uses a special area of memory called the stack, where the contents of specified registers or memory locations can be temporarily saved.
- It is a top-down data structure whose elements are accessed using the pointer that is implemented using the SP and SS registers.
- As we go on storing the data words onto the stack, the pointer goes on decrementing.

Declaring a STACK:

STACK is declared at the beginning of the program:

```
01 .MODEL SMALL
02
03 .STACK 100H ;a pseudo-instruction that tells
04              ;the assembler to allocate 256 bytes
05              ;(100h in hexadecimal) of memory for
06              ;the stack segment.
07 .DATA
08
09 ; declare variables here
10
11 .CODE
12 MAIN PROC
```

This portion of code will create a stack with 100 memory bytes. In each push or pop operation, the stack pointer will decrease or increase by 2.

B. Hour 2

Discussion: PUSH and POP OPERATIONS

PUSH OPERATIONS

There are mainly two operations, PUSH and PUSHF, in the stack for push operations. In case of the PUSH command, the new value is added at the top of the stack.

PUSH: Used to add a new source (16-bit register or memory word) to the stack.

Syntax:

***PUSH* source**

i.e, PUSH AX; this pushes the content of the AX register to the stack

Execution of PUSH causes:

a. SP is decreased by 2

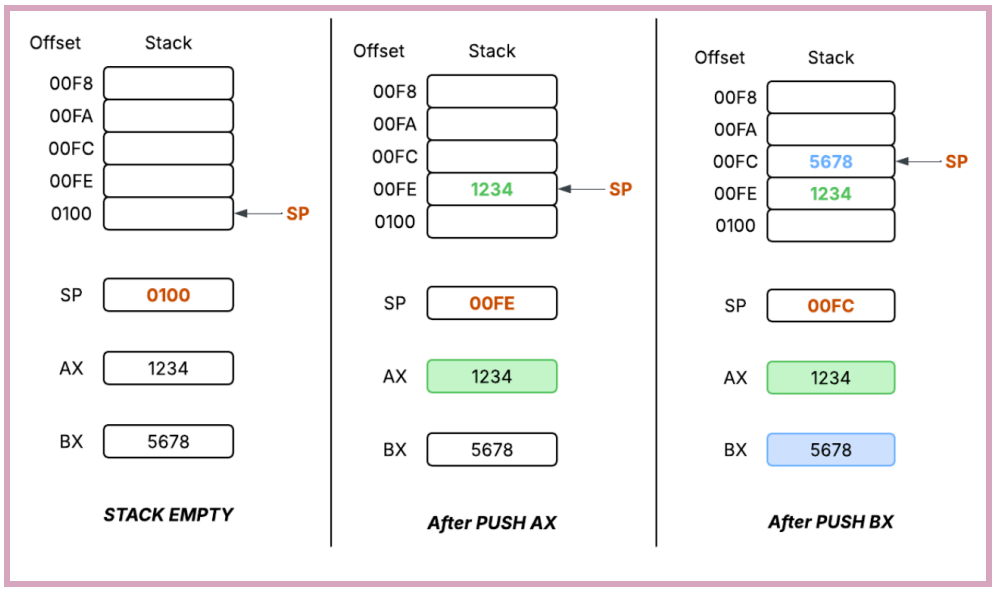
b. A copy of the source contents is moved to the address specified by SS: SP.

c. Source remains unchanged.

PUSHF: has no operand and it pushes the contents of the FLAG register onto the stack. Usually, this is done whenever the processor is interrupted.

Syntax: *PUSHF*

In the case of ***PUSHF***, all the flags are stored in the stack. This is mainly used when any procedure is called. In such a case, the current condition of the program is saved into to stack by the **PUSHF** command. This is automatically called when any procedure or macro is called inside the program.



PUSH OPERATION IN STACK

POP OPERATIONS

There are mainly two operations, POP and POPF, in the stack for push operations. In case of the POP command, the top element of the stack is removed from the stack and stored in the destination register.

POP is used to remove the top item from the stack to the destination (i.e., a 16-bit register or memory word).

Syntax:

***POP* destination**

i.e, POP BX; this will pop the top element of the stack and save it to the BX register.

Execution of POP causes:

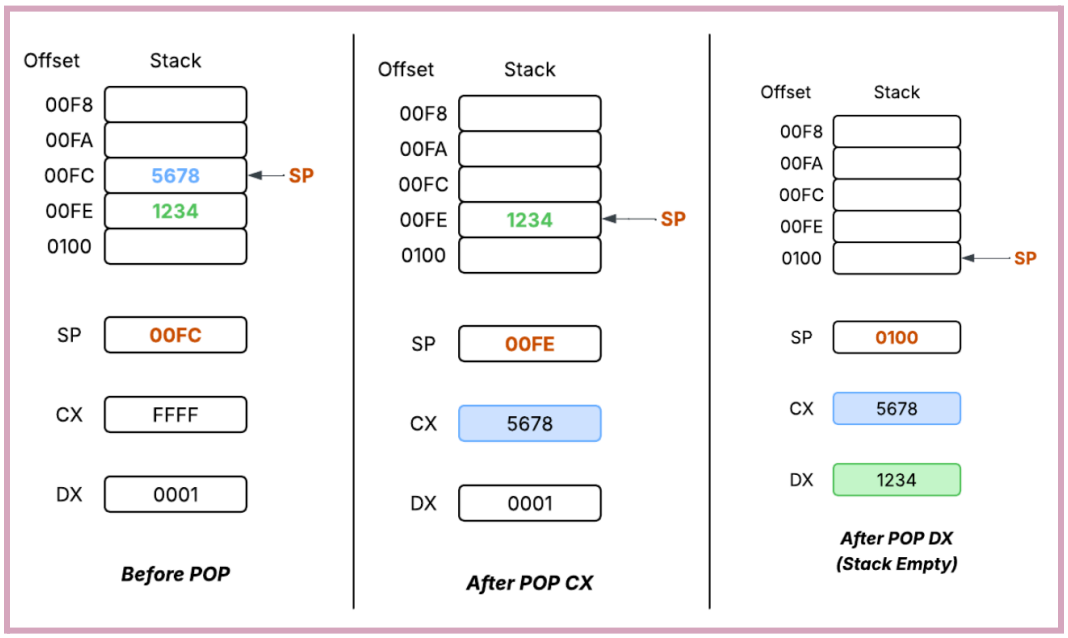
a. The contents of SS: SP (top of the stack) are moved to the destination.

b. SP is increased by 2

POPF has no operand and pops the top of the stack into the FLAG register. SP is incremented by 2 after executing this instruction.

Syntax: ***POPF***

POPF is just the opposite of **PUSHF**. This command restores all the flags to the flag register. Thus, the previous state of the system is restored. This is automatically called while returning from a procedure or macro.



POP OPERATION IN STACK

EXAMPLE CODE:

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

MOV AX,@DATA
MOV DS,AX

    MOV AX, 122d      ; move 'z' to AX (ASCII of 'z' is 122)
    PUSH AX           ; push 'z' to stack
    MOV AX, 49d       ; move '1' to AX (ASCII of '1' is 49)
    PUSH AX           ; push '1' to stack

    POP BX            ; pop '1' (top element) to BX
    MOV DX, BX        ; output content of BX
    MOV AH, 2
    INT 21h

    POP BX            ; pop 'z' (next top element) to BX
    MOV DX, BX        ; output content of BX
    MOV AH, 2
    INT 21h

MOV AX,4C00H
INT 21H

MAIN ENDP
    END MAIN
```

Use of a stack in a function call:

STACK is internally used during any procedure call.

1. When any procedure is called, the current memory location, values of all registers, and local variables are pushed onto the stack. This is done internally.
2. When returning from procedures, the saved elements that were pushed onto the stack during calling the procedure are popped from the stack. And in this way, the execution can start from the previous point.

When to use stacks:

- Temporary save area for registers
- To save the return address for CALL
- To pass arguments
- Local variables
- Applications which have LIFO nature such as Applications which have LIFO nature, such as reversing a string

HOUR 3: PROBLEM SOLVING

PROBLEMS:

1. Swap two numbers using only push and pop instructions.
2. Store five numbers in a stack and find the sum while popping them.
3. Input a string and check whether all the letters of the word are unique or not.
4. Take five elements in an array and print them in reverse order using a stack.
5. Input a number in a register. Print the number in reverse order using a stack.
6. Input a string and reverse it using a stack.
7. Input a word and check whether the word is a palindrome or not.
8. Input a number. Find all the factors of that number and store them in a stack.
9. You are given an array of 5 integers. Using a stack, find the maximum, minimum, and average values. Push all elements onto the stack, then pop them one by one to calculate these values. Finally, display the results.
10. Use Stack to sort 3 numbers in ascending[or descending] order.

11. Input a number. Starting from 1 to that number, find all the prime numbers.
12. Write a program that lets the user type some text, consisting of words separated by blanks, ending with a carriage return, and displays the text in the same word order as entered, but with the letters ·in each word reversed. For example, "this is a test" becomes

    "siht si a tset".

13. Write an assembly language program using EMU8086 that takes an integer n as input and prints a right-angled triangle pattern consisting of stars (*). The number of lines in the triangle should be equal to the input number n, and each line should contain a number of stars corresponding to the line number, starting from 1 star on the first line up to n stars on the n-th line. **[YOU CAN USE ONLY IMPLICIT LOOP; MUST HAVE TO USE STACK TO STORE CX VALUE]**

Sample Input	Sample Output
Enter the height of the triangle: 5	* * * * * * * * * *

14. In a customer service center, incoming support tickets are logged by agents and stored in a system temporarily using a stack. Each new ticket is added to the top of the stack. At the end of the day, a supervisor decides how many recent tickets need to be reviewed and pops them from the stack to process them in reverse order of arrival (most recent first).

    Write an 8086 Assembly language program to simulate this system. First, ask the user how many support ticket numbers they want to log (input), and push each of those numbers onto the stack. Then, ask how many ticket numbers need to be reviewed (popped), and pop and display that many numbers from the stack in the order they are removed.

Sample Input	Sample Output
Enter the number of support tickets to log: 3 Enter the support ticket number: 3 Enter the support ticket number: 4 Enter the support ticket number: 5 Enter the number of tickets to review: 2	Reviewing ticket number: 5 Reviewing ticket number: 4